Обработка ошибок времени выполнения в PostgreSQL

Фролков Иван, Postgres Professional





Ошибки

- Это неизбежно
- Более того, некоторые ошибки – совершенно нормальное и нередко даже желаемое явление
- Ошибки надо уметь обрабатывать



Варианты обработки ошибок

- Просто выводим сообщение “ой, что-то случилось”
  - Для интерактивных приложений еще туда-сюда
  - Иногда даже и правильно
- Ничего не делаем – все падает как падает
  - Не так уж плохо
- Оба варианта выше вполне жизнеспособны в ряде случаев
- Или, что хуже, просто игнорируем (*when others then null*)
  - Для backend-a такое поведение - просто катастрофа, надо ругаться в мониторинг или хотя бы сбрасывать в лог и **прерывать работу**



Не “кто виноват”, не “что случилось”, а “что делать”

- Ошибки СУБД классифицирует как “что сломалось” или даже “где сломалось”, а вот в приложении их надо классфицировать как “что случилось с БД/приложением” и, исходя из этого, “что делать дальше”
- Что случилось с
  - БД – все нормально/сомнительно/**плохо**/**совсем плохо**
  - Приложением – все нормально/сомнительно/**плохо**/**совсем плохо**
- А что делать-то? В общем случае:
  - **Не обращать внимания (предупреждения)**
    - **Транзакция продолжает выполняться**
  - **Сообщить пользователю/внешней системе/отправить в файл ошибок (constraints)**
    - **Откат транзакции/savepoint**
  - **Повторить операцию сразу (deadlock_detected, serialization_failure)**
    - **Откат транзакции, сигнализировать в лог, при частых повторах – в мониторинг**
  - **Повторить операцию через какое-то время (проблемы с соединением, недоступность бизнес-сущностей – нет товара на складе, например)**
    - **Откат транзации, сигнализировать в монтиторинг**
  - **Сделать что-то, специфичное для приложения, и повторить операцию**
    - **Откат транзакции, что-то сделать, сигнализировать в мониторинг**
  - **Остановить приложение или его часть**
    - **Откат транзакции, сигнализировать в мониторинг**
  - **Остановить все/переключить**
    - **Откат транзакции, полная остановка, сигнализировать в мониторинг**



Анатомия ошибки

- Postgres
  - SEVERITY (DEBUG, LOG, INFO, NOTICE, WARNING и EXCEPTION)
  - **SQLSTATE**
  - message, hint, detail, column, table, schema, constraint, datatype



Например

```
work=# \set VERBOSITY verbose
work=# create table ct(id int primary key,
 val text check(val ~ '^\d+'));
CREATE TABLE
work=# insert into ct values(1,'zz');
ERROR:  23514: new row for relation "ct" violates check constraint "ct_val_check"
DETAIL:  Failing row contains (1, zz).
SCHEMA NAME:  public
TABLE NAME:  ct
CONSTRAINT NAME:  ct_val_check
LOCATION:  ExecConstraints, execMain.c:2021
```



SQLSTATE

- **23514 – код ошибки (check_violation)**
  - **23** – класс ошибок (иногда довольно странный)
  - **514** – условие в классе
- **in_failed_sql_transaction** – название ошибки
  - Не у всех ошибок могут быть названия
  - ***Appendix A. PostgreSQL Error Codes***
- **0, 1, 2, 3, 4, A, B, C, D, E, F и G –** стандартные, **5-9** и **I-Z** – расширения
    - И для классов, и для условий
  - `ZQ001` – “товар временно отсутствует”



Дополнительные классы ошибок

- IM – ODBC
- Y? – клиенты Postgres
  - YE001 – ECPG “out of memory”



01, 02 – Warning/No Data

- В зависимости от контекста может быть совершенно нормальным
- 01 – в pg только privilege_not_granted/revoked
- 02 – мне попадался только в ECPG, я им не пользуюсь
  - select … into strict в plpgsql почему-то выбрасывает P0002



03,09,0A...

- Самый длинный список
- Например:
  - 27000 triggered_data_change_violation

    tuple to be deleted was already modified by an operation triggered by the current command
  - 21000 cardinality_violation

    more than one row returned by a subquery used as an expression
  - 39004 null_value_not_allowed

    lower bound of FOR loop cannot be null
- Некоторые ошибки (HV00N – fdw_unable_to_establish_connection) можно обрабатывать как класс 08



03,09,0A,22...

- Скорее всего это ошибки приложения или его конфигурации и дальнейшее выполнение операции может привести к неверным результатам. Не исключено, что дальнейшая работа невозможна.
- Сообщаем пользователю/внешней системе/оператору о возникшей проблеме
- **Приложение находится в некорректном состоянии. Не исключено, что дальнейшая работа невозможна**
- **БД после возникновения ошибки остается в корректном состоянии**



22 – Data Exception

- array_subscript_error, invalid_argument_for_logarithm, invalid_escape_sequence, invalid_row_count_in_result_offset_clause, invalid_xml_processing_instruction и т.д.
  - **Ошибки приложения**
  - **БД в корректном состоянии**



23 - Integrity Constraint Violation

- FOREIGN KEY, CHECK, UNIQUE, EXCLUDE
  - Это ошибка проведения операции и свидетельствует о том, что БД не удалось перевести в некорректное состояние. Дальнейшая работа возможна.
  - Можно определить, где возникла ошибка (constraint, table, schema в ошибке) и указать пользователю/внешней системе/оператору, исходя из этих данных, в чем именно проблема.
- **Ошибки приложения**
- **БД в корректном состоянии**



42 Syntax Error or Access Rule Violation

- syntax_error, insufficient_privilege, datatype_mismatch, ambiguous_column и пр.
  - **Тоже ошибки приложения**
  - **БД в корректном состоянии**



Не все ошибки одинаковы

- 01, 02, 09 и др. – редко
- 0B, 0F и др. – не встречается
- 0Z002 – только get stacked diagnostics вне exception
- И т.д.
- 40 – фактически в pg используются 40001 (serialization_failure) и 40P01 (deadlock_detected)



08, 40001, 53, 40P01, 72000

- 08 – Connection Exception, 40001 – serialization_failure, 53 - Insufficient Resources, 40P01 - deadlock_detected, 72000 – snapshot_too_old
  - Возможно, требуется повторение операции
  - В случае повтора необходимо указать таймаут
- **Приложение в рабочем состоянии**
- **БД находится в корректном состоянии, но, возможно, неработоспособна**



58, XX

- 58000 system_error
  - 58030 io_error
  - 58P01 undefined_file
  - 58P02 duplicate_file
- XX000 internal_error
  - XX001 data_corrupted
  - XX002 index_corrupted
- **Приложение находится в рабочем состоянии**
- **БД находится в нерабочем состоянии. Дальнейшая работа невозможна.**

Повторы операции. Deadlock

- Транзакция **А** захватила строку **а** и пытается захватить строку **б**, транзакция **Б** захватила строку **б** и пытается захватить строку а (и т.д. в случе трех и более транзакций)
- На самом деле дедлок – невозможность сериализовать (преобразовать в последовательность) некоторое множество транзакций.
- Дедлоки могут возникнуть всегда. Даже если у вас сейчас его нет, в будущем всегда может появиться
- Транзакции с дедлоками необходимо повторять



Повторы операций. Deadlock

- Повтор может быть не один
- Более того, повтор может случиться несколько тысяч раз!
- Надо быть к этому готовым: **выполнять нетранзакционные операции непосредственно перед коммитом при отсутствии deferred триггеров**.
  - Параноики могут использовать распределенные транзакции или идемпотентных участников (тем более, что это полезно и без дедлоков)
- Недостаток Postgres – нельзя задать стратегию выбора deadlock victim.
  - Правда, никто и не жаловался (ну, кроме меня)



Повторы операций. Deadlock

- Можно ли полностью избавиться от дедлоков?
  - Нет
  - Можно снизить вероятность их возникновения



Отступим от темы

- Deadlocks in connection pool
  - Если каждому треду надо больше одного соединения
  - min pool size = T*(C-1)+1, где
    - T – максимальное число тредов
    - C – максимальное число одновременных соединений у треда
  - Могут быть разные пулы для разных серверов
  - Проблема – раздутые connection pool ведут к снижению производительности
  - Проблема – размер connection pool должен соответствовать max_connections, чтобы не попать на deadlock уже именно сессий



Повторы операций. serialization_failure

- Нарушение уровня изоляции serialization_failure
- Совпадает с обработкой дедлоков
- Для read-only транзакций можно избежать, см. документацию
  - Правда, почти всегда будет достаточно repeatable read



Повторы операций. Класс 08 – Connection Exception

- За исключением 08P01 protocol_violation ничего особо интересного – надо просто повторять попытки соединения
  - Отличие от deadlock_detected и serialization_failure – повтор не транзакции, а попытки получить соединение
- 08P01 – ошибка, скорее всего, невосстановимая и потребуется вмешательство
- **Приложение в рабочем состоянии (если не 08P01); возможно, некорректно сконфигурировано; требуется отражение проблемы в мониторинге**
- **БД в корректном состоянии**



Повторы операций, классы 53, 54

- Это Insufficient Resourses и Program Limit Exceeded
- 53 подходит для ошибок приложения типа “Все операторы заняты, попробуйте позже” и т.п.
  - Оговорка – возможно, для бизнес-ошибки “недостаточно ресурсов” можно использовать свой класс
- Возможно, приложение не сможет продолжить работу
- **БД в корректном состоянии; тем не менее, она, возможно, неработоспособна**



Повторы операций.
Класс 72 Snapshot Failure
72000 – Snapshot too old

- Ошибка специфична для Postgres
- Возникает, когда сессия пытается обратиться к странице, которая уже была очищена.
- Параметр old_snapshot_threshold
  – См. документацию.
- Возможно, потребуется вмешательство.
- **БД в корректном состоянии**



Повтор операций. Таймауты между повторами

- Для deadlock_detected и serialization_failure – минимальный
  - Возможно, через несколько попыток есть смысл подождать побольше
- Для 08, 53, 54 – побольше
- Если долго не удается выполнить операцию – сильно ругаться в мониторинг



20

- Некоторый курьез
- 20000 case_not_found
  - Он такой один бог весть почему
  - Ну и слава богу: некоторая неортогональность указывает на жизненность



Выводы

- Необходимо построить закрытый список ошибок, которые допускают повторное выполнение (deadlock_detected, serialization_failure, некоторые из классов 08, ошибка snapshot_too_old, какие-нибудь специфичные для приложения 53??? и т.д.)
- Все ошибки класса 23 – сообщения о неудачных попытках перевести БД в некорректное состояние, иногда вполне ожидаемые (ннапример, unique_violation)
- Все ошибки, кроме ошибок классов 00,01,02,08, требуют отката транзакции
  - Некоторые ошибки могут быть откачены до savepoint
- Все ошибки, кроме классов 00,01,02,23, отчасти 08, ошибок deadlock_detected, serialization_failure, snapshot_too_old, специфичных для приложения, говорят об ошибках либо в приложении, либо в конфигурации приложения
- Все ошибки и предупреждения надо выводить в лог и мониторинг
- Все остальное – не только в лог и мониторинг, для backend-процессов лучше всего прервать выполнение



Классы ошибок (Postgres) **Работаем**, **Показываем ошибку**, **Повторяем**, **Авария**, **Катастрофа**

- 00 – Successful Completion - **все нормально**
- 01 – Warning, 02 – No Data (тоже Warning) (только в ECPG)
  - ***отображение в логах и, возможно, в мониторинге (вообще в pg штука достаточно экзотическая)***
- 03,09,0A,0B,0F, 0L,0P,0Z,20,21,22,24,25,26,27,28,2B,2D,2F,34,38,39,3B,3D,3F,42,44,54,55,58,F0,HV,P0 – No Data, Feature not Supported, Data Exception, Syntax Error и пр.
  - ***ошибка приложения или его конфигурации в том или ином виде, отображение в мониторинге, отмена выполняемом операции (не только транзакции), возможно, отключение подсистемы или всего приложения***
- 23 – Integrity Constraint Violation
  - ***Для интерактивных приложений - ошибка данных: неверный ввод. Отображение в мониторинге только в неожиданных случаях; для пакетных заданий – аварийная остановка или запись в файл ошибочных данных***
- 08 – Connection Exception, 53 - Insufficient Resources
  - *Попытки переподключения с каким-то разумным интервалом*
- **40**001 – serialization_failure, **40**P01 - deadlock_detected, 72000 – snapshot_too_old
  - ***Возможно повторение операции сразу же, отображение в мониторинге и логах***
- 57 – Operator Intervention
  - ***Требуется вмешательство – отображение в мониториге и логах, возможно, остановка приложения, как минимум отмена операции***
- 58 (io_error и пр.), XX – Internal Error (internal_error, data_corrupted, index_corrupted) или бизнес-ошибка mission-critical уровня
  - ***Срочная остановка всех процессов или, возможно, promote реплики до мастера***



Обработка ошибок в plpgsql

```
begin
…
exception
   when deadlock_detected then…
   when unique_violation then…
   when ‘2201B’  then -- invalid regex
   when ‘23000’ then -- получаем целый класс
end
```

- Неявный savepoint
  - Много savepoint (>64 в сессии) ведет к снижению производительности (не так критично в PostgresPro)
- When others then raise sqtate ‘…..’ или просто raise;
  - Обработка ошибок в plpgsql обычно имеет смысл
    - Для отката
    - Для изменения ошибки на что-то имеющее смысл для приложения:

```
when  unique_violation then
 raise ‘53U01’ using message=’Все операторы заняты’
```



Обработка ошибок в plpgsql

- GET STACKED DIAGNOSTICS

```
DECLARE
  text_var1 text;
  text_var2 text;
  text_var3 text;
BEGIN
  -- здесь происходит обработка, которая может вызвать
исключение
  …
EXCEPTION WHEN OTHERS THEN
  GET STACKED DIAGNOSTICS text_var1 = MESSAGE_TEXT,
                          text_var2 = PG_EXCEPTION_DETAIL,
                          text_var3 = PG_EXCEPTION_HINT;
END;
```



Обработка ошибок в plpgsql

- PG_EXCEPTION_CONTEXT
  - Стек вызовов

```
GET STACKED DIAGNOSTICS
   stack=PG_EXCEPTION_CONTEXT;
```



Java

- JDBC4: SQLException
  - BatchUpdateException
  - RowSetWarning – напрямую к СУБД не относится
  - SerialException – serialize/deserialize errors
  - SQLClientInfoException - “when one or more client info properties could not be set on a Connection”
  - SQLNonTransientException
    - SQLDataException - класс 22
    - SQLIntegrityConstraintViolationException – класс 23
    - SQLSyntaxErrorException – класс 42
    - SQLInvalidAuthorizationSpecException – класс 28
    - SQLFeatureNotSupportedException – класс 0A – в Postgres сейчас такого класса нет
    - SQLNonTransientConnectionException – класс 08
  - SQLRecoverableException - ...*failed operation might be able to succeed if the application performs some recovery steps and retries the entire transaction or in the case of a distributed transaction, the transaction branch. At a minimum, the recovery operation must include closing the current connection and getting a new connection...*
  - SQLTransientException
    - SQLTransactionRollbackException – класс 40
    - SQLTransientConnectionException – класс 08
  - SQLWarning
  - SyncFactoryException – работа с RowSet
  - SyncProviderException – работа с RowSet



- java.sql.**SQLException** (implements java.lang.Iterable<T>)
  - java.sql.**BatchUpdateException**
  - java.sql.**SQLClientInfoException**
  - java.sql.**SQLNonTransientException**
    - java.sql.**SQLDataException**
    - java.sql.**SQLFeatureNotSupportedException**
    - java.sql.**SQLIntegrityConstraintViolationException**
    - java.sql.**SQLInvalidAuthorizationSpecException**
    - java.sql.**SQLNonTransientConnectionException**
    - java.sql.**SQLSyntaxErrorException**
  - java.sql.**SQLRecoverableException**
  - java.sql.**SQLTransientException**
    - java.sql.**SQLTimeoutException**
    - java.sql.**SQLTransactionRollbackException**
    - java.sql.**SQLTransientConnectionException**
  - java.sql.**SQLWarning**
    - java.sql.**DataTruncation**



pgjdbc,pgjdbc-ng

- Это все не очень-то и важно: pgjdbc выбрасывает только SQLException
  - Тем не менее никто не жаловался
- Зато pgjdbc-ng умеет
  SQLIntegrityConstraintViolationException – класс 23000
  PGSQLSimpleException – все остальное (ну вот так получилось)
  - Но не умеет hint



Spring

- Иное представление о прекрасном
  - Самобытная иерархия
  - https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/dao/package-tree.html
- Соединениями обычно управляет connection pool, так что класс 08 увидеть не получится
  - SingleConnectionDatasource
  - Но это может быть как org.springframework.jdbc.CannotGetJdbcConnectionException, так и org.springframework.transaction.CannotCreateTransactionException
    - getCause() все равно 08001



- org.springframework.dao.**DataAccessException**
    - org.springframework.dao.**NonTransientDataAccessException**
        - org.springframework.dao.**CleanupFailureDataAccessException**
        - org.springframework.dao.**DataIntegrityViolationException**
            - org.springframework.dao.**DuplicateKeyException**
        - org.springframework.dao.**DataRetrievalFailureException**
            - org.springframework.dao.**IncorrectResultSizeDataAccessException**
                - org.springframework.dao.**EmptyResultDataAccessException**
        - org.springframework.dao.**InvalidDataAccessApiUsageException**
        - org.springframework.dao.**InvalidDataAccessResourceUsageException**
            - org.springframework.dao.**IncorrectUpdateSemanticsDataAccessException**
            - org.springframework.dao.**TypeMismatchDataAccessException**
        - org.springframework.dao.**NonTransientDataAccessResourceException**
            - org.springframework.dao.**DataAccessResourceFailureException**
        - org.springframework.dao.**PermissionDeniedDataAccessException**
        - org.springframework.dao.**UncategorizedDataAccessException**
    - org.springframework.dao.**RecoverableDataAccessException**
    - org.springframework.dao.**TransientDataAccessException**
        - org.springframework.dao.**ConcurrencyFailureException**
            - org.springframework.dao.**OptimisticLockingFailureException**
            - org.springframework.dao.**PessimisticLockingFailureException**
                - org.springframework.dao.**CannotAcquireLockException**
                - org.springframework.dao.**CannotSerializeTransactionException**
                - org.springframework.dao.**DeadlockLoserDataAccessException**
        - org.springframework.dao.**QueryTimeoutException**
        - org.springframework.dao.**TransientDataAccessResourceException**

Spring framework - Postgres

- SQLErrorCodeSQLExceptionTranslator
  - BadSqlGrammarException **03000,42000,42601,42602,42622,42804,42P01**
  - InvalidResultSetAccessException
  - DataIntegrityViolationException **23000,23502,23503,23514**
    - DuplicateKeyException **23505**
  - PermissionDeniedDataAccessException
  - DataAccessResourceFailureException **53000,53100,53200,53300**
  - TransientDataAccessResourceException
  - CannotAcquireLockException **55P03**
  - DeadlockLoserDataAccessException **40P01**
  - CannotSerializeTransactionException **40001**



Spring framework

- SQLExceptionTranslator
  - Можно установить свой, преобразующий SQLException в что-то более подходящее для предметной области
- sql-error-codes.xml
- К Postgres это уже имеет достаточно слабое отношение



Hibernate

- `public String getSQLState()`

- class org.hibernate.HibernateException
    - class org.hibernate.JDBCException
        - class org.hibernate.exception.ConstraintViolationException
        - class org.hibernate.exception.DataException
        - class org.hibernate.exception.GenericJDBCException
        - class org.hibernate.exception.JDBCConnectionException
        - class org.hibernate.exception.LockAcquisitionException
        - class org.hibernate.exception.SQLGrammarException



JPA

- getCause()
  - javax.persistence.**PersistenceException**
    - javax.persistence.**EntityExistsException**
    - javax.persistence.**EntityNotFoundException**
    - javax.persistence.**LockTimeoutException**
    - javax.persistence.**NonUniqueResultException**
    - javax.persistence.**NoResultException**
    - javax.persistence.**OptimisticLockException**
    - javax.persistence.**PessimisticLockException**
    - javax.persistence.**QueryTimeoutException**
    - javax.persistence.**RollbackException**
    - javax.persistence.**TransactionRequiredException**



Hibernate+JPA

- “However, if we're using Hibernate as a JPA persistence provider, these exceptions may get wrapped into PersistenceException.”
- JDBC+Spring+Hibernate+JPA...



Java - выводы

“...наряду с мужчиной и женщиной появились двужчина, дваба и два вспомогательных пола — уложники и поддержанки. Жизнь, особенно эротическая, ... усложнилась до крайности.“ (Станислав Лем, “Путешествие двадцать первое”)

- Такое цветущее разнообразие наводит на две мысли:
  - Это реальная проблема
  - Которую не так-то просто решить
- Что делать?
  - Пользоваться существующим: для небольших задач это вполне удобно
  - SQLSTATE



C#

- Я не разбираюсь в C#, но...
  - Видимо, нет такого буйства мысли, как в Java
  - Сами, все сами
  - Класс DbException имеет свойство SqlState
- npgsql
  - PostgresException имеет свойство SqlState (естественно)
  - Есть messageText, hint, detail и т.д.



PHP, Go

“На все прочие явления жизни мастерская штемпелей и печатей отозвалась только одной синей табличкой: «Дежурная няня»”.

- Все очень сдержанно.
- Ошибка и ошибка
- SQLSTATE



PHP

- PDO:
```
$dbh->setAttribute(
    PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION
);
```
  - PDOException
  - Доступен только SQLSTATE, остальное придется получать из сообщения об ошибке
- pgsql
  - Исключений нет, надо получать pg_result_error/pg_result_error_field, там можно получить hint, details, constraint и т.п.



Go

- До SQLSTATE и прочего можно добраться
- Ну и хватит



Python/psycopg2

- https://www.psycopg.org/docs/errors.html
  - pgcode – это на самом деле SQLSTATE
    - “String representing the error code returned by the backend, None if not available.”
-
```
StandardError
|__Warning
|__Error
   |__InterfaceError
   |__DatabaseError
      |__DataError
      |__OperationalError
      |__IntegrityError
      |__InternalError
      |__ProgrammingError
      |__NotSupportedError
```
- Почему-то не сочли нужным выделять Transient & Recoverable: 08 – это DatabaseError, а 40P01 -OperationalError, вместе с 53,54,57,HV



Выводы

- Или разнообразные иерархии ошибок даже в пределах одной экосистемы, или все очень минималистично
- Все иерархии отвечают на главный вопрос – “кто виноват?”, а надо ответ на “что делать?”
- **Самый разумный вариант – SQLSTATE**



Выбрасываем ошибки

```
RAISE [ уровень ] 'формат' [, выражение [, ... ]]
[ USING параметр = значение [, ... ] ];
RAISE [ уровень ] имя_условия [ USING параметр =
выражение [, ... ] ];
RAISE [ уровень ] SQLSTATE 'sqlstate' [ USING параметр
= выражение [, ... ] ];
RAISE [ уровень ] USING параметр = выражение [, ... ];
RAISE ;
```

- Уровень - DEBUG, LOG, INFO, NOTICE, WARNING и EXCEPTION
  - Уровень – отдельно, SQLSTATE - отдельно



А что выбрасывать-то?

- Не надо изобретать многочисленные классы ошибок, не хватает только двух – “**бизнес-ресурс временно недоступен**” и “**ошибка misson-critical уровня**”
- Для неудачных проверок – 23, для неверной логики – 22, для “Пока нет денег для операции”, “Товар отсутствует на складе”, “Все операторы заняты”, “Неожиданный баланс пользователя” или “Обнаружен несанкционированный вывод средств” - см. предыдущий пункт, для “глобальные проблемы с железом/софтом, все укладываем[, переключаемся на реплику]” – 58
- Кроме того, есть hint, details и проч.



Выводы

- Обработкой ошибок мало кто занимается
  - Дедлоки и т.п. не оставляют вариантов
- В Postgres в общем-то есть все что надо
- В Java в процессе упрощения сделали еще сложнее
- Не в Java этим вопросом себя особо не изнуряли – может, и слава богу.
- SQLSTATE!



Вопросы?

